Une nouvelle technique de filtrage basée sur la décomposition de sous-réseaux de contraintes

Philippe Jégou **Cyril Terrioux**

LSIS - UMR CNRS 6168
Université Paul Cézanne (Aix-Marseille 3)
Avenue Escadrille Normandie-Niemen 13397 Marseille Cedex 20, France
{philippe.jegou, cyril.terrioux}@univ-cezanne.fr

Résumé

Dans ce papier, nous introduisons une nouvelle technique de filtrage pour les réseaux de contraintes. Elle est basée sur une propriété appelée *cohérence structurelle*. Il s'agit d'une cohérence paramétrable que nous noterons w-*SC*. Cette cohérence est basée sur une approche significativement différente de celles en usage. Alors que les cohérences classiques s'appuient généralement sur des propriétés locales étendues à l'ensemble du réseau, cette cohérence partielle considère à l'opposé la cohérence globale de sous-problèmes. Ces sous-problèmes sont définis par des graphes de contraintes partiels dont la largeur arborescente est bornée par une constante w, qui correspond au paramètre associé à la cohérence. Nous introduisons un algorithme de filtrage qui permet d'obtenir la w-SC cohérence. Cette cohérence est ensuite analysée pour la positionner par rapport aux cohérences classiquement utilisées dans les CSP. Cette étude montre que cette nouvelle cohérence est généralement incomparable avec celles figurant dans la littérature. Enfin, nous présentons des résultats expérimentaux préliminaires pour évaluer l'utilité de cette approche.

1 Introduction

Un CSP (Constraint Satisfaction Problem [16]), parfois également appelé réseau de contraintes, est exprimé par la donnée d'un ensemble fini de variables X, qui doivent être affectées dans leurs domaines (finis) de valeurs donnés par D, et une solution doit satisfaire un ensemble fini de contraintes C. Pour résoudre un CSP, les approches usuelles sont basées sur les algorithmes de type backtracking, dont la complexité en temps est de l'ordre de $O(e.d^n)$ où n est le nombre de variables, e est le nombre de contraintes et d est la taille maximum des domaines. Pour résoudre efficacement un CSP, ces algorithmes exploitent en général des techniques de filtrages, soit avant la recherche sous forme de pré-traitements, soit durant la recherche. La qualité de l'outil de filtrage utilisé joue alors un rôle crucial au niveau de l'efficacité de la recherche.

Les effets d'une méthode de filtrage consistent généralement en la suppression de valeurs dans les domaines. Ces valeurs peuvent être supprimées car nous avons la garantie qu'elles ne peuvent figurer dans aucune solution (elles sont dites incohérentes). Ainsi, les filtrages sont-ils basés sur la notion de *cohérence*.

Dans la mesure où la suppression de toutes les valeurs incohérentes est généralement irréaliste d'un point de vue pratique (c'est un problème NP-difficile), les filtrages sont basés sur des propriétés de cohérences moins fortes, appelées *cohérences partielles*. Ainsi, une valeur peut satisfaire une cohérence partielle, même si elle n'apparaît dans aucune solution (elle est globalement incohérente). A l'opposé, d'autres valeurs peuvent contredire la cohérence partielle considérée, et ainsi être retirées des domaines, sans altérer la satisfiabilité d'un CSP. Les cohérences partielles qui ont été définies jusqu'à présent [2] sont généralement définies par des propriétés de *cohérence locale* dont l'exploitation est étendue à l'ensemble du réseau de contraintes considéré. Par exemple, pour satisfaire la cohérence d'arc (AC) qui est la cohérence partielle la plus utilisée, une valeur doit posséder au moins une valeur compatible (appelée *support*) dans le domaine des variables figurant dans son voisinage au niveau du réseau de contraintes. Sinon, cette valeur est retirée de son domaine (filtrée) et cette suppression peut conduire à d'autres suppressions dans les domaines des variables voisines. En utilisant un mécanisme appelé *propagation de contraintes*, la première suppression peut être

étendue à l'ensemble du réseau. Ainsi, les cohérences partielles correspondent généralement à des propriétés locales qui doivent finalement être vérifiées sur l'ensemble du réseau. D'un point de vue pratique, l'intérêt d'une cohérence partielle est lié à son pouvoir de filtrage ainsi qu'au coût de sa réalisation (en termes de complexité en temps et en espace).

Dans cette contribution, nous introduisons une nouvelle forme de cohérence partielle qui doit satisfaire principalement deux critères, en particulier sur les jeux de données (instances) les plus difficiles à résoudre :

- l'efficacité pratique,
- la puissance de filtrage.

Pour cela, cette nouvelle cohérence devra être :

- paramétrable, de sorte à contrôler la complexité du filtrage, et donc *a priori*, son efficacité pratique,
- adaptée à la structure du réseau de contraintes, en exploitant ses sous-structures,
- adaptée à la dureté des contraintes, en privilégiant l'exploitation des contraintes les plus difficiles à satisfaire.

Cette nouvelle cohérence est appelée *w-SC* car il s'agit d'une cohérence paramétrée (w est le paramètre considéré) qui est basée sur des propriétés Structurelles du réseau.

Elle est définie en considérant une relaxation du CSP (i.e. un sous-problème) qui sera constituée par un réseau de contraintes partiel dont la largeur arborescente (sa tree-width) est bornée par une constante w [15]. Nous avons choisi ce type de sous-problèmes car leur traitement peut s'appuyer sur des algorithmes dont la complexité en temps est polynomiale en w, mais aussi du fait des progrès observés ces dernières années au niveau des techniques dites de décompositions, qui peuvent désormais les résoudre de façon extrêmement efficace d'un point de vue pratique (voir par exemple [13]). De plus, alors que les cohérences partielles classiques considèrent les contraintes sans véritablement tenir compte de leur dureté, ici, nous aurons la possibilité de prendre en compte celle-ci assez naturellement en sélectionnant le sous-problème servant au filtrage de sorte à ce que les contraintes les plus dures y figurent.

Plus précisément, étant donné un sous-réseau de contraintes correspondant à un graphe partiel de largeur arborescente bornée, nous dirons qu'une valeur vérifie la w-SC cohérence si celle-ci figure au moins dans une solution de ce sous-problème.

Il s'agit donc d'une approche fondée sur une idée similaire à la notion de *cohérence inverse* [11], même si formellement, elle est de nature différente.

Ainsi, cette nouvelle cohérence nous permet de définir un nouveau type de filtrage qui s'avère finalement très différent de ceux en usage dans les CSP. Notamment, nous montrerons que la w-SC cohérence est incomparable avec les méthodes reconnues parmi les plus efficaces dans le domaine, comme AC ou SAC [7]. En particulier, les expériences montreront que le comportement de la w-SC cohérence est radicalement différent de celui d'AC ou de SAC, étant notamment plus efficace pour les jeux de données difficiles, tout en s'avérant moins efficace sur des jeux de données faciles (problèmes sous-contraints).

D'un point de vue opérationnel, nous introduisons ici un algorithme de filtrage dont la complexité en temps est $O(n^2.w.d^{w+2})$.

Ce papier est organisé comme suit. La section suivante rappelle les notions classiques de cohérences partielles et les filtrages qui leur sont associés. Dans la section 3, nous introduisons la w-SC cohérence et le filtrage qui lui est associé. La section 4 présente une analyse théorique sur les relations qui peuvent se présenter entre la w-SC cohérence et les autres propriétés de cohérence alors que la section 5 fournit une analyse expérimentale de ce filtrage. Enfin, la section 6 présente les suites qui pourront être données à ce travail.

2 Notions de base

2.1 Notations

Un *problème de satisfaction de contraintes à domaines finis* ou *réseau de contraintes fini* (X, D, C) est défini par la donnée d'un ensemble de variables $X = \{x_1, \dots x_n\}$, d'un ensemble de domaines $D = \{D(x_1), \dots D(x_n)\}$ (le domaine $D(x_i)$ contient les valeurs possibles pour la variable x_i), et d'un ensemble de contraintes C portant sur les variables. Une contrainte $c_i \in C$ est définie d'une part par sa portée $S_C(c_i)$ et d'autre part par une relation de compatibilité associée $R_C(c_i)$. La portée d'une contrainte est un sous-ensemble ordonné de variables, c'est-à-dire $S_C(c_i) = (x_{i_1}, x_{i_2}, \dots x_{i_{a_i}})$ où a_i est appelée *arité* de la contrainte c_i. La relation $R_C(c_i) \subseteq D(x_{i_1}) \times D(x_{i_2}) \cdots \times D(x_{i_{a_i}})$ définit les combinaisons de valeurs compatibles pour les variables de $S_C(c_i)$, chaque combinaison de valeurs compatibles permettant de satisfaire la contrainte. Ici, nous noterons par S_C l'ensemble des portées de contraintes, à savoir $S_C = \{S_C(c_1), S_C(c_2) \dots S_C(c_e)\}$ où $e = |C|$ est le nombre de contraintes. Une solution de (X, D, C) est une affectation de l'ensemble des variables qui satisfait l'ensemble des contraintes. Sans manque de généralité, nous supposerons que chaque variable apparaît une fois au moins dans la portée d'une contrainte. Si toutes les contraintes d'un CSP sont binaires (i.e. elles portent sur deux variables exactement), alors la structure de ce réseau binaire (appelé dans ce cas CSP binaire) peut être représentée par un graphe (X, S_C)

appelé *graphe de contraintes*.

Nous supposerons que les relations ne sont pas vides et peuvent être représentées par des tables comme dans le cadre de la Théorie des Bases de Données Relationnelles. De plus, et sans manque de généralité, nous supposerons que le réseau de contraintes est connexe et normalisé (deux contraintes différentes ont des portées différentes) et pour simplifier le propos, nous ne considérerons ici que les réseaux binaires. Une contrainte c_k telle que $S_C(c_k) = (x_i, x_j)$, c_k sera alors notée c_{ij}. En général, les CSP sont résolus en utilisant des algorithmes de type backtracking qui peuvent être réellement efficaces en pratique, notamment s'ils exploitent judicieusement des méthodes de filtrage avant et/ou pendant la recherche, afin d'éviter l'exploration de zones de l'espace de recherche inutiles. Ces filtrages sont formellement basés sur des notions dites de cohérences partielles.

2.2 Cohérences partielles

La cohérence partielle la plus ancienne et si l'on peut dire, la plus populaire, est appelée *cohérence d'arc* (AC). Etant donné un CSP $P = (X, D, C)$, une valeur $v_i \in D(x_i)$ vérifie la cohérence d'arc vis-à-vis de la contrainte $c_{ij} \in C$ si et seulement s'il existe une valeur valide $v_j \in D(x_j)$ telle que $(v_i, v_j) \in R_C(c_{ij})$. On dit alors que v_j est un support de v_i pour la contrainte c_{ij}. Une valeur $v_i \in D(x_i)$ vérifie la cohérence d'arc si v_i vérifie la cohérence d'arc vis-à-vis de chaque contrainte $c_{ij} \in C$. Un domaine $D(x_i)$ vérifie la cohérence d'arc sur c_{ij} si et seulement si $\forall v_i \in D(x_i)$, la valeur v_i vérifie la cohérence d'arc, et le CSP $P = (X, D, C)$ vérifie la cohérence d'arc si et seulement si $\forall D(x_i) \in D$, le domaine $D(x_i)$ vérifie la cohérence d'arc pour toute contrainte $c_{ij} \in C$. Un filtrage des domaines basé sur AC revient à supprimer les valeurs qui ne vérifient pas la cohérence d'arc. Quand une valeur v_i est supprimée, un mécanisme appelé *propagation de contraintes* est mis en œuvre de sorte à supprimer les valeurs dont l'unique support était la valeur supprimée v_i (aucune autre valeur de $D(x_i)$ n'est compatible avec elle), et ce processus peut être étendu à toutes les autres valeurs, sous les mêmes conditions. Pour les réseaux binaires, plusieurs algorithmes très efficaces ont été proposés comme, par exemple, AC-2001 [5] ou AC3rm [14]. La meilleure complexité en temps obtenue est $O(e.d^2)$. Ce type d'algorithmes est véritablement efficace en pratique, à tel point que le filtrage peut également être mis en œuvre au sein d'un algorithme de recherche de solutions. Néanmoins, le pouvoir de filtrage d'AC se révèle parfois limité du fait de l'aspect très local inhérent à la définition de la cohérence associée. Aussi, des propriétés de cohérence plus forte, et qui permettent donc l'obtention de filtrages plus puissants ont donc été définies. Notamment, [9] a introduit la notion de *k-cohérence* qui est définie en considérant des sous-ensembles de k variables. Pour la k-cohérence, une nouvelle contrainte d'arité égale à $k-1$ est ajoutée au réseau lors du filtrage à partir du moment où une affectation pourtant cohérente sur $k-1$ variables ne peut être étendue à une $k^{ème}$ variable. Si le réseau vérifie la i-cohérence, pour $2 \leq i \leq k$, le CSP est dit *fortement k-cohérent*. A titre de rappel, la cohérence appelée *Strong-PC* aussi dite *cohérence forte de chemin* correspond à la *forte 3-cohérence*. Plus la valeur de k est élevée, plus la puissance du filtrage est forte. Malheureusement, la complexité en temps et en espace est alors $O(n^k.d^k)$ [6] et ce type de filtrage recèle ainsi des inconvénients considérables. Du fait de cette complexité, ces filtrages se révèlent généralement inutilisables, même pour de petites valeurs de k ($k = 3$ s'avère souvent irréaliste en pratique). De plus, ce type de filtrage va rajouter de nouvelles contraintes dans le réseau, dont l'arité est de l'ordre de $k-1$, et pour lesquelles l'espace requis se révèle ainsi très rapidement prohibitif, même pour des petites valeurs de k. Afin d'éviter de tels problèmes, principalement le second induit par l'ajout de contraintes additionnelles, d'autres propriétés de cohérences partielles ont été proposées dans la littérature. Par exemple, [11] a proposé la *k-inverse cohérence*. Le filtrage associé supprime les valeurs qui ne peuvent être étendues à $k-1$ variables pour former une affectation cohérente. Pour ce filtrage, plus puissant qu'AC, le problème relatif à la complexité en espace n'est certes plus présent, mais le problème de la complexité en temps demeure dans la mesure où celle-ci est de l'ordre de $O(n^k.d^k)$. Aussi, ces auteurs ont-ils naturellement suggéré de limiter l'exploitation de cette cohérence à de petites valeurs de k. Dans [11], une autre sorte de cohérence inverse, qui est définie dans un esprit similaire, a été introduite. Elle est appelée *NIC* pour *neighborhood-inverse consistency* (cohérence de voisinage inverse). Ici, le filtrage d'un domaine est induit par la compatibilité des valeurs associées aux variables par rapport au sous-problème défini par son voisinage dans le réseau. De fait, la complexité est alors liée au degré maximum Δ d'une variable dans le graphe de contraintes, et l'algorithme qui a été proposé possède donc une complexité en $O(\Delta^2.(n + e.d).d^{\Delta+1})$.

Une autre façon pour définir des cohérences partielles est fondée sur la prise en compte de sous-problèmes induits par des affectations de la forme $x_i = v_i$, en testant alors leur cohérence partielle. Par exemple, un CSP est dit *Singleton arc-cohérent* (ce qui est noté *SAC*) [7] si pour tous les domaines et pour toutes leurs valeurs $v_i \in D(x_i)$, le sous-problème induit par l'affectation $x_i = v_i$ vérifie la cohérence d'arc sur les sous-domaines. La complexité en temps

du meilleur algorithme connu pour le filtrage associé est $O(e.n.d^3)$ [4].

Pour conclure ce rapide panorama sur les cohérences partielles et filtrages associés, nous devons rappeler que les cohérences partielles généralement considérées comme étant les plus intéressantes d'un point de vue pratique en termes de filtrages sont AC ou SAC qui semblent receler le meilleur compromis entre le coût en temps (et donc l'efficacité pratique) et la puissance de filtrage.

3 La Cohérence Structurelle

3.1 La w-SC Cohérence

La Cohérence Structurelle est basée sur la notion de graphe partiel dont la largeur arborescente (*tree-width*) est bornée par une constante w. La largeur arborescente d'un graphe se définit à partir de la notion de décomposition arborescente (*tree-decomposition*) qui fut introduite formellement dans [15]. Elle a déjà été exploitée dans le domaine des CSP notamment pour définir des classes d'instances polynomiales [12] mais aussi, et surtout, pour proposer des méthodes de résolution particulièrement efficaces pour résoudre des réseaux de contraintes, sous la réserve que ceux-ci possèdent des propriétés topologiques intéressantes [8, 13].

Notre objectif ici s'avère cependant différent de ces précédents travaux puisque nous allons exploiter ces notions essentiellement afin de définir des cohérences partielles. Pour cela, nous introduisons la notion de *w-PST* (pour *partial spanning tree-decomposition*) qui consiste en un graphe partiel dont la décomposition arborescente est de largeur w. Avant cela, nous rappelons la définition de décomposition arborescente :

Définition 1 *Une décomposition arborescente d'un graphe $G = (V, E)$ est une paire (N, T) où $T = (I, F)$ est un arbre avec un ensemble de nœuds I et d'arêtes F, et $N = \{N_i : i \in I\}$ une famille de sous-ensembles de V, telle que chaque sous-ensemble (appelé cluster) N_i est un nœud de T et vérifie :*
(i) $\cup_{i \in I} N_i = V$,
(ii) $\forall \{x, y\} \in E$, $\exists i \in I$ avec $\{x, y\} \subseteq N_i$, et
(iii) $\forall i, j, k \in I$, si k est sur le chemin entre i et j dans T, alors $N_i \cap N_j \subseteq N_k$.

La largeur w d'une décomposition arborescente (N, T) est égale à $max_{i \in I}|N_i| - 1$. La largeur arborescente w^* de G est la largeur minimale sur toutes ses décompositions arborescentes.

Nous définissons maintenant des graphes partiels possédant une largeur arborescente donnée.

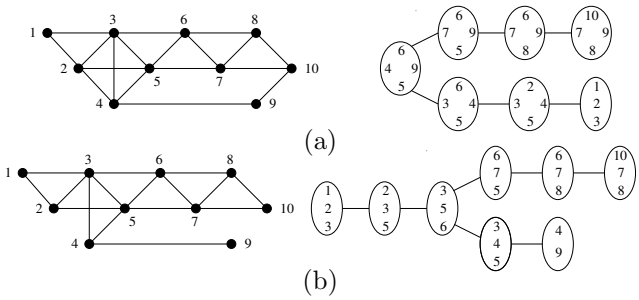


FIG. 1 – (a) Un graphe de largeur arborescente 3 et la décomposition arborescente correspondante. (b) Un 2-PST du graphe donné en (a), et une décomposition arborescente.

Définition 2 *Etant donné un graphe $G = (V, E)$, un graphe partiel de G est un graphe $G' = (V, E')$ où $E' \subseteq E$. Une décomposition arborescente partielle recouvrante de largeur w pour G, notée w-PST, est un graphe partiel de G dont la largeur arborescente est w.*

Le graphe donné dans la figure 1(a) est un 3-PST parce que sa largeur arborescente vaut 3 (une décomposition arborescente optimale est donnée dans cette figure). Dans la figure 1, nous avons le graphe partiel induit par la suppression des arêtes $\{2, 4\}$ et $\{9, 10\}$ dans le graphe donné dans la figure 1(a). Il s'agit d'un 2-PST car sa largeur arborescente vaut 2 comme permet de s'en assurer la décomposition arborescente optimale donnée dans cette figure.

Nous introduisons maintenant la notion de sous-problème d'un CSP induit par l'affectation d'une variable.

Définition 3 *Etant donné un CSP $P = (X, D, C)$, une variable $x_i \in X$ et une valeur $v_i \in D(x_i)$, le sous-problème de P induit par l'affectation $x_i = v_i$ est $P|_{x_i = v_i} = (X, D', C')$ où $D'(x_i) = \{v_i\}$ et pour tout $j \neq i, D'(x_j) = D(x_j)$ et $C' = C$, excepté pour les relations associées aux contraintes incluant x_i qui sont restreintes aux tuples où figure la valeur v_i.*

Avant de définir la cohérence structurelle basée sur un w-PST, nous devons introduire la notion de problème relaxé qui est défini par un sous-ensemble de contraintes d'un CSP donné.

Définition 4 *Etant donné un CSP $P = (X, D, C)$ et $W \subseteq S_C$, le problème relaxé de P induit par W est le CSP $P(W) = (X, D, C')$ où $W = S_{C'}$.*

Pour définir la w-SC cohérence, le problème relaxé est défini par un sous-ensemble de contraintes formant une décomposition arborescente partielle recouvrante de largeur w.

Algorithme 1: Comp-w-SC (**In** : (X, W) : Graph ; **InOut** : $P = (X, D, C)$: CSP)

```
1 for x_i ∈ X do
2     D'(x_i) ← ∅;
3 for x_i ∈ X do
4     for v_i ∈ D(x_i) do
5         if v_i ∉ D'(x_i) then
6             if Solution(P(W)|_{x_i=v_i}, Sol) then
7                 for v_j ∈ Sol do
8                     D'(x_j) ← D'(x_j) ∪ {v_j}
9             else D(x_i) ← D(x_i) − {v_i}
```

Définition 5 *Etant donné un CSP $P = (X, D, C)$ et un w-PST $G = (X, W)$ de (X, S_C) :*

- *La valeur $v_i \in D(x_i)$ est w-SC-cohérente par rapport à G si et seulement si $P(W)|_{x_i=v_i}$ possède une solution.*
- *Le domaine $D(x_i)$ est w-SC-cohérent par rapport à G si et seulement si $\forall v_i \in D(x_i)$, la valeur v_i est w-SC-cohérente par rapport à G.*
- *Le CSP $P = (X, D, C)$ est w-SC-cohérent par rapport à G si et seulement si $\forall D(x_i) \in D$, $D(x_i)$ est w-SC-cohérent par rapport à G.*

Il faut noter que cette définition de cohérence est associée à la notion d'affectation de variable à des valeurs de domaines. Elle peut aisément être généralisée à des affectations plus larges, qui porteraient sur des sous-ensembles de variables. Par manque de place, et pour simplifier cette présentation, nous limiterons la définition de w-SC-cohérence à l'affectation d'une seule variable.

3.2 Filtrage associé

Comme il est d'usage classiquement pour les cohérences dans les CSP, le filtrage associé à la w-SC-cohérence consiste tout simplement à supprimer les valeurs qui ne satisfont pas la propriété correspondante.

Définition 6 *Etant donné un CSP $P = (X, D, C)$ et un w-PST $G = (X, W)$ de (X, S_C), le CSP filtré par l'exploitation de la w-SC-cohérence est le CSP noté $w\text{-}SC(P, W) = (X, D', C')$ où :*

- *$D' = \{D'(x_1), \dots D'(x_n)\}$ où $\forall D'(x_i) \in D'$, $D'(x_i) = \{v_i \in D(x_i) : v_i$ est w-SC-cohérente par rapport à $G\}$.*
- *$S_{C'} = S_C$.*
- *$\forall c'_{ij} \in C', R_{C'}(c'_{ij}) = R_C(c_{ij}) \cap D'(x_i) \times D'(x_j)$.*

Notons qu'étant donné un CSP et un w-PST (X, W) de (X, S_C), $w\text{-}SC(P, W)$ est unique. De plus, pour s'assurer que la w-SC-cohérence définit un filtrage valide, nous devons nous assurer qu'aucune valeur filtrée n'apparaît dans une solution du CSP considéré. C'est nécessairement le cas puisque les valeurs éliminées n'apparaissent dans aucune solution du CSP relaxé considéré. Nous présentons maintenant l'algorithme appelé *Comp-w-SC* qui réalise le filtrage w-SC correspondant. Contrairement aux algorithmes de filtrage classiques, comme notamment ceux réalisant AC, l'obtention de cette cohérence partielle ne nécessite pas le recours à des mécanismes de propagation une fois qu'une suppression de valeur a été réalisée. En effet, alors que les algorithmes classiques réalisent des suppressions, puis propagent celles-ci, dans *Comp-w-SC*, une fois qu'une valeur v_i a été validée en obtenant une solution dans laquelle elle figure, elle ne pourra plus être supprimée et ainsi v_i se trouve définitivement validée. Ceci est tout simplement dû au fait que la valeur v_i apparaît dans une solution du CSP relaxé, en compagnie d'autres valeurs (que l'on peut assimiler aux supports de la valeur v_i). De plus, puisque ces valeurs figurent dans une solution, elles ne pourront plus être supprimées car ces valeurs se trouvent elles aussi validées et pour elles, il ne sera pas nécessaire de tester leur w-SC cohérence.

Dans l'algorithme *Comp-w-SC*, la fonction *Solution($P(W)|_{x_i=v_i}$, Sol)* est appelée pour réaliser le test de cohérence. Si v_i apparaît dans une solution $Sol = (v_1, v_2, \dots v_i, \dots v_n)$ de $P(W)$, la fonction renvoie *true* et Sol constitue l'autre résultat de cet appel. Dans le cas contraire, la valeur *false* est renvoyée. Dans *Comp-w-SC*, D' correspond à l'ensemble des domaines contenant les valeurs déjà validées et donc mémorisées pendant la phase de filtrage. Ainsi, si une valeur v_i est déjà présente dans $D'(x_i)$, c'est tout simplement parce que cette valeur figure déjà dans une solution calculée et il ne sera donc pas nécessaire de tester ultérieurement sa w-SC-cohérence. Notons enfin qu'à la fin de l'exécution de *Comp-w-SC*, pour toutes les variables x_i, nous avons $D(x_i) = D'(x_i)$.

Théorème 1 *La complexité en temps de Comp-w-SC est $O(n^2.w.d^{w+2})$ alors que sa complexité en espace est $O(n.w.d^s)$ avec s la taille du plus grand séparateur dans le w-PST.*

Preuve : La fonction *Solution($P(W)|_{x_i=v_i}$)* est appelée au plus $n.d$ fois et le coût d'un appel à cette fonction est borné par $n.w.d^{w+1}$. En effet, elle peut être implémentée en utilisant des algorithmes de résolution basés sur les décompositions arborescentes de CSP comme TC [8] notamment, ou encore, pour être plus efficace en pratique comme BTD [13].

De plus, nous savons que la complexité en espace d'une méthode de résolution par décomposition comme BTD est relative à la taille des séparateurs entre les clusters de la décomposition arborescente.

Aussi, si s est la taille du plus grand séparateur dans le w-PST, comme le nombre de séparateurs est majoré par $n-1$, la complexité en espace se trouve bornée par $O(n.w.d^s)$. □

4 Relations avec les autres cohérences

Pour évaluer la puissance de filtrage de w-SC, nous présentons ici une analyse qui est développée dans un esprit similaire à celui qui a guidé l'analyse présentée dans [7] qui fournit une comparaison entre plusieurs cohérences locales. Ici, nous considérerons AC, SAC, Strong-PC, et plus généralement, la k-cohérence forte ainsi que la k-inverse cohérence. Ce type de comparaisons est basé sur l'existence de relations formelles entre cohérences dont nous rappelons le principe. Nous dirons qu'une cohérence CO_1 est *plus forte* qu'une cohérence CO_2 (ce sera noté $CO_2 \leq CO_1$) si pour tout CSP P pour lequel CO_1 est vérifiée, alors CO_2 est également vérifiée. Ainsi, tout algorithme obtenant CO_1 supprimera au moins les valeurs qui seront supprimées en utilisant CO_2. Nous dirons qu'une cohérence CO_1 est *strictement plus forte* qu'une cohérence CO_2 (ce sera noté $CO_2 < CO_1$) si $CO_2 \leq CO_1$ et s'il existe au moins un CSP P pour lequel CO_2 est vérifiée alors que CO_1 ne l'est pas. Notons par ailleurs que ces relations sont naturellement transitives. Finalement, nous dirons que CO_1 et CO_2 sont *incomparables* si aucune relation entre elles n'est vérifiée.

Théorème 2 *1-SC < AC et pour $w > 1$, w-SC et AC sont incomparables.*

Preuve : Il est clair qu'un 1-PST correspond exactement à un arbre (sous l'hypothèse de connexité). Aussi, puisque dans un réseau de contraintes constitué par un arbre, une valeur apparaît dans une solution si et seulement si elle vérifie AC, 1-SC ne peut filtrer plus de valeurs que ne le fait la cohérence d'arc, qui pour sa part va considérer l'ensemble des contraintes du problème. Par conséquent, nous avons déjà 1-SC $\leq$ AC. De plus, puisque d'autres valeurs du réseau peuvent être supprimées par AC via l'exploitation de contraintes qui n'apparaissent pas dans le 1-PST considéré, nous avons plus précisément 1-SC < AC.

Maintenant, si nous considérons w-SC et AC avec $w > 1$, ces deux cohérences s'avèrent incomparables. En effet, il suffit de constater que pour AC, une valeur du domaine d'une variable est supprimée si elle ne posséde pas de support pour une contrainte qui n'apparaît pas dans un w-PST. Dans ce cas, cette valeur sera peut être conservée par un filtrage de type w-SC. Inversement, une valeur peut être supprimée par un filtrage de type w-SC mais pas par AC. En effet, si nous considérons un CSP constitué de trois variables x_1, x_2 et x_3 ayant toutes le même domaine avec deux valeurs et de trois contraintes de différence c_{12}, c_{13} et c_{23}, AC ne filtre aucune valeur alors que 2-SC appliqué sur l'ensemble du problème (qui est un 2-PST) va supprimer toutes les valeurs. □

Théorème 3 *1-SC < SAC et pour $w > 1$, w-SC et SAC sont incomparables.*

Preuve : Puisque 1-SC < AC et puisque AC < SAC, par transitivité de <, nous avons 1-SC < SAC.

Par ailleurs, puisque SAC considère toutes les contraintes qui apparaissent dans le réseau, nécessairement, le filtrage peut supprimer les valeurs qui n'ont pas été supprimées par 2-SC. Inversement, si on considère l'exemple (c) présenté en page 216 de [7] et qui est un 2-PST satisfaisant SAC, nous pouvons facilement voir que 2-SC supprimera une valeur. Par conséquent, 2-SC et SAC sont incomparables. Ainsi, plus généralement, w-SC et SAC sont incomparables.□

Avant d'établir d'éventuelles relations avec des cohérences plus fortes, nous établissons les relations entre différents niveaux de w-SC cohérence. Pour les relations entre w-SC avec différentes valeurs de w, nous supposerons prendre en compte des CSP qui possèdent des décompositions arborescentes partielles recouvrantes de largeurs différentes telles que w-PST $\subseteq$ $(w+1)$-PST (i.e. les arêtes du w-PST considéré sont incluses dans celles du $(w+1)$-PST). Sans cela, nous ne disposerions d'aucune garantie sur la comparison entre w-SC et $(w+1)$-SC.

Théorème 4 *Si w-PST $\subseteq$ $(w+1)$-PST, alors w-SC < $(w+1)$-SC.*

Preuve : Si w-PST $\subseteq$ $(w+1)$-PST, les valeurs supprimées par w-SC cohérence en considérant le w-PST sont nécessairement supprimées en considérant le $(w+1)$-PST. De plus, puisqu'il existe des contraintes dans le $(w+1)$-PST qui ne figurent pas dans le w-PST, des valeurs supplémentaires seront détruites par la $(w+1)$-SC cohérence en considérant le $(w+1)$-PST. Par conséquent, nous avons w-SC < $(w+1)$-SC. □

Sur ces bases, et en appliquant le même principe que dans le théorème 2, nous établissons la propriété suivante qui peut être vue comme sa généralisation :

Théorème 5 *k-SC < $(k+1)$-cohérence forte.*

Preuve : Il est facile de voir que la largeur [10] d'un w-PST est exactement w. Par conséquent,

en appliquant les résultats présentés dans [10] et qui mettent en relation la largeur d'un réseau de contraintes avec le niveau de cohérence forte vérifiée, nécessairement, chaque valeur qui apparaît dans un domaine vérifiant la $(k+1)$-cohérence forte appartient à une solution de ce k-PST. Ainsi, elle ne sera pas supprimée par l'obtention de la k-SC cohérence. En outre, puisque d'autres valeurs de ce réseau peuvent être supprimées par la $(k+1)$-cohérence forte en exploitant des contraintes qui n'apparaissent pas dans le w-PST (avec $w = k$), il est facile de voir que la $(k+1)$-cohérence forte peut supprimer plus de valeurs que le filtrage par k-SC cohérence, et par conséquent, nous avons bien k-SC $<$ $(k+1)$-cohérence forte. □

Théorème 6 *Pour $k > 2$, la k-cohérence forte et k-SC sont incomparables.*

Preuve : Pour montrer que pour $k > 2$, k-SC et la k-cohérence forte sont incomparables, il est alors suffisant de montrer que certaines valeurs peuvent être supprimées par k-SC alors qu'elles ne sont pas filtrées par la k-cohérence forte. Considérons le CSP correspondant au problème de la k-coloration d'un graphe complet à $k+1$ sommets. Ce réseau est un k-PST mais il n'admet pas de solution. Par conséquent, l'obtention de la k-SC cohérence sur ce réseau supprimera toutes les valeurs. Par contre, si maintenant nous considérons l'application de la k-cohérence forte, aucune valeur ne sera supprimée. Par ailleurs, comme la k-cohérence forte prend en compte plus de contraintes qu'il n'en apparaît dans un k-PST, des valeurs seraient supprimées par la k-cohérence forte alors qu'elles ne peuvent l'être par k-SC. Par conséquent, pour $w > 2$, k-SC et la k-cohérence forte sont incomparables. □

Corollaire 1 *2-SC $<$ Strong-PC.*

Finalement, notons que nous pouvons tout à fait remplacer la k-cohérence forte par la k-inverse-cohérence [11] dans l'énoncé du théorème 4. Il est également facile d'établir que NIC et w-SC (pour $w > 2$) sont incomparables.

La figure 2 synthétise les relations entre cohérences usuelles. Les flèches représentent les relations de type $>$ alors que les lignes pointillées indiquent que les deux cohérences concernées sont incomparables.

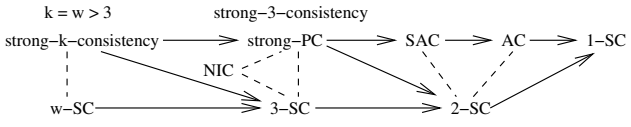


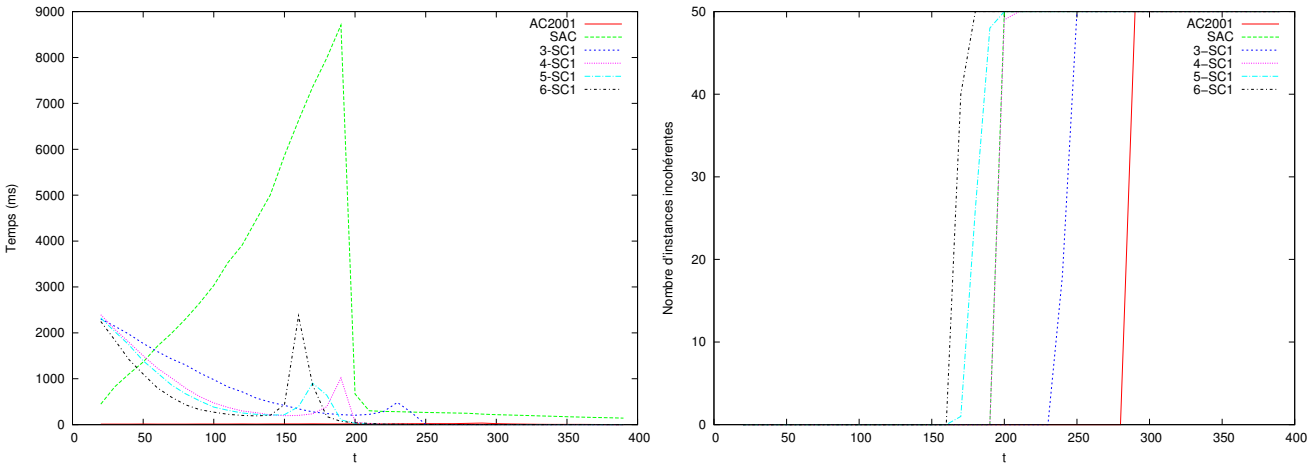
FIG. 2 – Relations entre cohérences

5 Expérimentations

5.1 Protocole expérimental

La qualité du filtrage réalisé par w-SC dépend fortement du w-PST considéré. Dans les résultats expérimentaux présentés ici, le calcul du w-PST est obtenu en utilisant une méthode heuristique qui s'appuie sur la notion de k-arbre [1]. L'exploitation des k-arbres se justifie ici car il s'agit de graphes, qui, pour une largeur arborescente donnée, possèdent le nombre maximum d'arêtes et devraient ainsi potentiellement contenir le nombre maximum de contraintes. Un graphe G est un k-arbre si G est le graphe complet à k sommets (il s'agit alors du k-arbre trivial) ou s'il existe un sommet x de degré k dont le voisinage est un graphe complet et si le graphe obtenu en supprimant de G le sommet x et toutes les arêtes qui lui sont incidentes est un k-arbre. Sur la base de cette définition inductive, on constate que les k-arbres peuvent être construits en commençant par un graphe à k sommets, et que pour chaque ajout de sommet x, celui-ci est connecté aux sommets d'une k-clique déjà présents dans le graphe précédent. Dans le cadre de nos expérimentations, et pour la méthode heuristique que nous utilisons, nous choisirons à chaque étape le sommet x qui minimise $\prod_{c_{ij}} t_{ij}$ où c_{ij} est une contrainte recouverte par la clique formée par x et les sommets de la k-clique considérée dans le graphe précédent et t_{ij} est le rapport entre le nombre de tuples interdits et le nombre total de tuples potentiels. t_{ij} est généralement appelée la dureté de la contrainte c_{ij}. La clique initiale sera par ailleurs choisie en utilisant un procédé similaire. Les choix opérés par cette heuristique sont donc motivés par la volonté d'obtenir des sous-problèmes les plus contraints possibles (cf. choix d'un maximum de contraintes de dureté maximum) de sorte à obtenir la puissance de filtrage la plus forte. Nous avons également implémenté une autre méthode qui consiste à calculer d'abord un k-arbre grâce à la méthode décrite précédemment puis à essayer d'ajouter heuristiquement autant de contraintes que possible tout en garantissant que la largeur arborescente ne dépasse pas w. Les contraintes sont considérées dans l'ordre décroissant de leur dureté. Dans les résultats fournies, nous noterons w-SC1 (respectivement w-SC2) les résultats obtenus lors de l'application de notre algorithme avec un w-PST construit selon la première méthode (respectivement la seconde). Dans les deux cas, nous exploitons un k-arbre avec $k = w$ et le sous-problème relatif au w-PST considéré est résolu grâce à la méthode de décomposition BTD [13].

Pour le filtrage AC, nous avons implémenté AC2001 [5]. Le choix de AC2001 pourrait se discuter au niveau expérimental si on tient compte du temps de cal-



(a) (b)

FIG. 3 – Résultats obtenus sur des instances aléatoires pour les classes (200,20,5970,t) : (a) temps d'exécution en millisecondes, (b) nombre d'instances incohérentes détectées.

cul dans la mesure où d'autres algorithmes peuvent se révéler plus rapides, mais, nous verrons qu'ici, cette question ne se pose finalement pas. Pour le filtrage SAC, nous avons utilisé une implémentation naïve de SAC. Tous les algorithmes sont implémentés en C. Notons que nous avons également comparé nos résultats avec une version plus élaborée de SAC (en l'occurrence SAC3 [3] fournie dans le solveur Java Abscon). Concernant le temps d'exécution, notre implémentation est parfois moins efficace que SAC3 mais cela ne change en rien la nature de la comparaison avec w-SC. Bien sûr, les deux versions de SAC ont exactement le même pouvoir de filtrage. Aussi, nous considérons uniquement notre version naïve dans les résultats fournis.

On pourrait, en première analyse, penser que ce type de filtrage est particulièrement adapté aux problèmes structurés (au sens d'une largeur arborescente de petite valeur). Cependant, ce n'est pas le cas, car, dans la mesure où w-SC travaille sur des graphes partiels, il ne semble pas nécessaire de se focaliser sur des graphes structurés. Les résultats expérimentaux montrent d'ailleurs que w-SC s'avère plus intéressant que SAC, par exemple, que ce soit en temps d'exécution ou en capacité de filtrage quand la densité s'accroît et que par conséquent la largeur arborescente augmente.

Ces trois algorithmes ont été comparés sur des instances aléatoires. Le générateur[1] considère 4 paramètres en entrée : n, d, e et t. Il génère des CSP de la classe (n, d, e, t) avec n variables dont la taille uniforme des domaines vaut d et qui possèdent e contraintes binaires ($0 \leq e \leq \frac{n(n-1)}{2}$) pour lesquelles t tuples sont interdits ($0 \leq t \leq d^2$). Les résultats présentés correspondent aux moyennes observées sur des jeux de 50 instances (ayant toutes des graphes de contraintes connexes) par classe. Les expérimentations ont été réalisées sur un PC sous Linux équipé d'un processeur Intel Pentium IV 3, 2 GHz et doté de 1 Go de mémoire.

[1] Voir la page http ://www.lirmm.fr/∼bessiere/generator.html

5.2 Comparaisons d'AC et SAC avec w-SC

Nous avons testé plusieurs classes d'instances obtenues en faisant varier le nombre de variables (jusqu'à 200 variables), la taille des domaines (jusqu'à 40 valeurs), la densité du graphe de contraintes et la dureté des contraintes. Toutefois, par manque de place, nous ne fournirons ici que les résultats obtenus en faisant variant t pour 200 variables, 20 valeurs par domaine et 5970 contraintes (densité de 30%) dans la figure 3 et les résultats obtenus sur quelques classes réprésentatives dans le tableau 1. Notons que nous ne fournissons pas dans la figure 3 les résultats obtenus par w-SC2 car ils sont très proches de ceux de w-SC1 (w-SC2 détecte simplement quelques instances supplémentaires comme étant incohérentes tout en demandant un temps d'exécution légèrement plus important).

Avant de comparer notre algorithme avec AC ou SAC, nous nous intéressons d'abord au choix d'une valeur convenable pour w. Dans la figure 3, si on considère le nombre d'instances détectées comme incohérentes par w-SC, nous pouvons constater que ce nombre augmente lorsque la valeur de w croît. Un tel résultat est prévisible dans la mesure où avec de plus grandes valeurs pour w, w-SC prend en compte plus de contraintes et est donc capable d'accomplir un fil-

Classes	AC			SAC			6-SC1			6-SC2		
(n,d,e,t)	tps	#inc	#suppr	tps	#inc	#suppr	tps	#inc	#suppr	tps	#inc	#suppr
(100,20,495,275)	1,8	0	9,24	198	50	104,68	70	20	486,82	441	48	79,20
(100,20,990,220)	2,4	0	0,22	11987	11	92,04	105	21	566,08	240	48	79,32
(100,20,1485,190)	3,4	0	0	4207	0	0,40	286	31	494,40	187	49	38,30
(100,40,495,1230)	4,6	0	0,92	3239	50	345,06	270	21	621,94	5709	48	106,76
(100,40,990,1030)	5,8	0	0	13229	0	0,08	515	30	809,34	4954	48	176,42
(100,40,1485,899)	8,2	0	0	11166	0	0	1622	32	902,14	1854	48	181,18
(200,10,1990,49)	2,6	0	20,96	88	50	38,28	128	22	350,62	72	48	56,36
(200,10,3980,35)	5,8	0	0,92	10503	49	261,74	248	0	34,86	637	0	249,56
(200,10,5970,30)	7,8	0	0,24	11335	0	11	423	0	54,62	708	2	241,34
(200,20,995,290)	4,6	0	57,58	224	50	65,52	190	26	670,32	7464	49	78,42
(200,20,1990,245)	6	0	3,36	3716	50	256,62	192	32	592,96	1109	50	20
(200,20,3980,195)	12,4	0	0,04	34871	0	1,82	573	25	808,46	592	49	70,48
(200,20,5970,165)	17	0	0	23307	0	0,04	2242	10	1179,88	1600	43	280,3

TAB. 1 – Temps d'exécution (en ms), nombre d'instances détectées comme incohérentes et nombre moyen de valeurs supprimées. Toutes les instances considérées n'ont aucune solution.

trage plus puissant. Ce phénomène est également vrai pour le temps qui augmente avec w. Selon nos observations, la valeur 6 pour w semble correspondre au meilleur compromis entre le temps d'exécution et la puissance de filtrage. D'une part, en exploitant des 6-PST, 6-SC1 (ou 6-SC2) prend en compte suffisamment de contraintes pour assurer un filtrage efficace. D'autre part, avec des valeurs de w plus grandes, le nombre de valeurs supprimées et donc le nombre d'instances incohérentes détectées n'augmentent guère tandis que le temps d'exécution et l'espace requis peuvent croître significativement par rapport à ceux observés pour $w = 6$. Ensuite, si nous comparons w-SC1 et w-SC2, nous pouvons observer dans le tableau 1 que généralement w-SC2 détecte plus d'instances incohérentes que w-SC1. Une fois de plus, un tel résultat est prévisible car w-SC2 exploite plus de contraintes que w-SC1. Pour la même raison, w-SC2 est souvent plus coûteux en temps que w-SC2.

Maintenant, si nous comparons le filtrage de w-SC avec celui de AC ou de SAC, nous pouvons noter dans la figure 3 que 3-SC détecte autant d'inconsistance que AC tandis que SAC se révèle tantôt meilleur, tantôt moins bon que w-SC selon la valeur de w. Néanmoins, nous pouvons observer que 6-SC détecte souvent plus d'instances incohérentes que SAC. Sur les 650 instances considérées dans le tableau 1, SAC est souvent meilleur que 6-SC1 (310 instances détectées comme incohérentes par SAC contre 270 par 6-SC1) mais moins bons que 6-SC2 (310 instances contre 530).

Concernant le temps d'exécution, selon la figure 3(a), AC est généralement plus rapide que w-SC, sauf pour les instances qui sont trivialement incohérentes. Dans ce dernier cas, w-SC parvient très souvent à détecter l'incohérence simplement en supprimant toutes les valeurs de la première variable considérée alors que AC effectue beaucoup de suppressions. Par rapport à SAC, w-SC est plus coûteux en temps sur les instances qui ne sont pas suffisamment dures (pour $t < 50$ sur la figure 3(a)) et donc trivialement cohérentes, car de nombreux appels à BTD sont requis. En revanche, quand la dureté est proche du seuil cohérent / incohérent ou au-delà, w-SC s'avère plus rapide que SAC. En effet, w-SC a besoin de supprimer moins de valeurs que SAC pour détecter l'incohérence car, par construction, il vérifie (et éventuellement supprime) chaque valeur d'une variable avant de traiter la variable suivante alors que dans AC ou SAC, les valeurs d'une variable donnée sont généralement supprimées à des moments différents (du fait du mécanisme de propagation). Ceci explique également les différences au niveau du nombre de valeurs filtrées entre 6-SC1 et 6-SC2 dans le tableau 1. Enfin, nous avons constaté que le comportement de w-SC par rapport à AC ou SAC s'améliore quand la densité du graphe de contraites augmente. Par conséquent, w-SC parvient à obtenir des temps d'exécution nettement meilleurs que SAC et à détecter plus d'instances incohérentes qu'AC et SAC dans la zone voisine du seuil cohérent / incohérent.

6 Discussion et conclusion

Nous avons proposé une nouvelle cohérence partielle paramétrée appelée w-SC-cohérence qui permet d'exploiter les propriétés internes d'un réseau de contraintes, à la fois au niveau structurel mais aussi au niveau de la dureté des contraintes. En l'état, cette nouvelle cohérence permet de définir des filtrages de domaines, à savoir la suppression de valeurs potentielles pour les variables. Cette nouvelle cohérence partielle diffère dans son approche de celles qui avaient été proposées dans la littérature jusqu'à présent. Ceci

peut se constater à la fois au niveau de sa comparaison théorique avec les cohérences partielles classiques mais également au niveau des premières expérimentations que nous avons réalisées. En effet, sur le plan théorique, le pouvoir de filtrage est en général incomparable avec les cohérences usuelles telles que la cohérence d'arc et ses généralisations mais aussi avec la cohérence SAC. Sur le plan pratique, nous avons pu également observer (non reporté dans la partie expérimentale) que ses capacités de filtrage font que ce ne sont pas les même valeurs qui sont supprimées et que les temps de calculs diffèrent au niveau des régions par rapport aux cohérences usuelles.

Il s'agit donc potentiellement d'une approche complémentaires à celles qui existaient précédemment.

Ceci conduit naturellement à proposer comme suite à ce travail, l'élaboration de propriétés de cohérences hybrides qui pourraient combiner w-SC avec d'autres cohérences telles que AC ou SAC, de sorte à fournir des filtrages à la fois plus robustes en termes de temps de calculs, mais aussi et surtout plus puissants en termes de pouvoir filtrant. Parmi les extensions potentielles à ce travail, w-SC pourrait également être étendue aux contraintes d'arité quelconque, ce qui ne semble pas particulièrement difficile d'un point de vue technique. Nous pouvons également généraliser w-SC en étendant le filtrage à des affectations partielles, par exemple en définissant le principe d'une k-w-SC cohérence qui produirait de nouvelle contraintes d'arité k. Dans les travaux qu'il faut également développer, même si 6-SC s'avère plus rapide et constitue un filtrage plus puissant que SAC notamment, du moins sur les jeux d'instances que nous avons testés, il serait utile de mieux identifier les bonnes valeurs du paramètre w. Comme cette cohérence s'appuie sur la notion de réseaux de contraintes partiels, formellement introduits ici en termes de graphes w-PST, une piste de recherche naturelle serait d'évaluer la densité requise des w-PST à considérer pour une valeur donnée de w. Une autre extension naturelle de ce travail consisterait également en l'intégration de cette nouvelle cohérence pour l'exploitation de son filtrage pendant une recherche, alors que nous avons focalisé notre travail pour le moment uniquement au niveau des pré-traitements. Ce type d'étude pourrait être entrepris tout d'abord dans le cadre des méthodes de résolution classiques basées sur le backtracking, mais aussi, et du fait de la proximité naturelle avec les méthodes fondées sur la décomposition de réseaux, sur les techniques à base justement de décomposition. Enfin, une étude approfondie devrait être menée sur le cadre général proposé dans [17] en étudiant différents types de sous-problèmes, pas seulement ceux liés aux décompositions arborescentes partielles recouvrantes, mais ce type de développement déborde du simple cadre de l'extension du travail sur la w-SC cohérence.

Références

[1] L. Beineke and R. Pippert. Properties and characterizations of k-trees. *Mathematika*, 18 :141–151, 1971.

[2] C. Bessiere. *Constraint Propagation*, chapter 3, pages 29–83. Handbook of Constraint Programming, F. Rossi, P. van Beek, T. Walsh, Elsevier, 2006.

[3] C. Bessière, S. Cardon, R. Debruyne, and C. Lecoutre. Efficient Algorithms for Singleton Arc Consistency. *Constraints*, 2010. A paraître.

[4] C. Bessière and R. Debruyne. Optimal and suboptimal singleton arc consistency algorithms. In *Proceedings of IJCAI*, pages 54–59, 2005.

[5] C. Bessière, JC. Régin, R.H.C. Yap, and Y. Zhang. An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2) :165–185, 2005.

[6] M.C Cooper. An optimal k-consistency algorithm. *Artificial Intelligence*, 41(1) :89–95, 1989.

[7] R. Debruyne and C. Bessière. Domain Filtering Consistencies. *JAIR*, 14 :205–230, 2001.

[8] R. Dechter. *Constraint processing*. Morgan Kaufmann Publishers, 2003.

[9] E. Freuder. Synthesizing constraint expressions. *CACM*, 21(11) :958–966, 1978.

[10] E. Freuder. A Sufficient Condition for Backtrack-Free Search. *JACM*, 29 (1) :24–32, 1982.

[11] E. Freuder and C. D. Elfe. Neighborhood inverse consistency preprocessing. In *Proceedings of AAAI*, pages 202–208, 1996.

[12] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence*, 124 :343–282, 2000.

[13] P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artificial Intelligence*, 146 :43–75, 2003.

[14] C. Lecoutre and F. Hemery. A study of residual supports in arc consistency. In *Proceedings of IJCAI*, pages 125–130, 2007.

[15] N. Robertson and P.D. Seymour. Graph minors II : Algorithmic aspects of treewidth. *Algorithms*, 7 :309–322, 1986.

[16] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier, 2006.

[17] G. Verfaillie, D. Martinez, and C. Bessière. A Generic Customizable Framework for Inverse Local Consistency. In *Proceedings of AAAI*, pages 169–174, 1999.